Closing The Performance Gap

William D. Gropp

Mathematics and Computer Science

www.mcs.anl.gov/~gropp

Performance Gap vs. Demo Gap

- But I saw a demo at Supercomputing!
- Clarke's Third law:
  - Any sufficiently advanced technology is indistinguishable from magic
- Demo gap
  - Corollary to Clarke's 3rd law:
    - Any sufficiently rigged demo is indistinguishable from magic
  - Gropp's conjecture
    - All supercomputing demos are sufficiently rigged

Real and Idealized Computer Architectures

- Any algorithm assumes an idealized architecture
  - Common choice:
    - Floating point work costs time
    - Data movement is free
  - Real systems:
    - Floating point is free (fully overlapped with other operations)
    - Data movement costs time…a *lot* of time
- Classical complexity analysis for numerical algorithms is *no longer correct* (more precisely, no longer *relevant*)
  - Known since at least BLAS2 and BLAS3

Sparse Matrix-Vector Product

- Common operation for optimal (in floating-point operations) solution of linear systems
- Sample code:

```
for row=1,n
    m   = i[row] - i[row-1];
    sum = 0;
    for k=1,m
        sum += *a++ * x[*j++];
    y[i] = sum;
```

- Data structures are a[nnz], j[nnz], i[n], x[n], y[n]

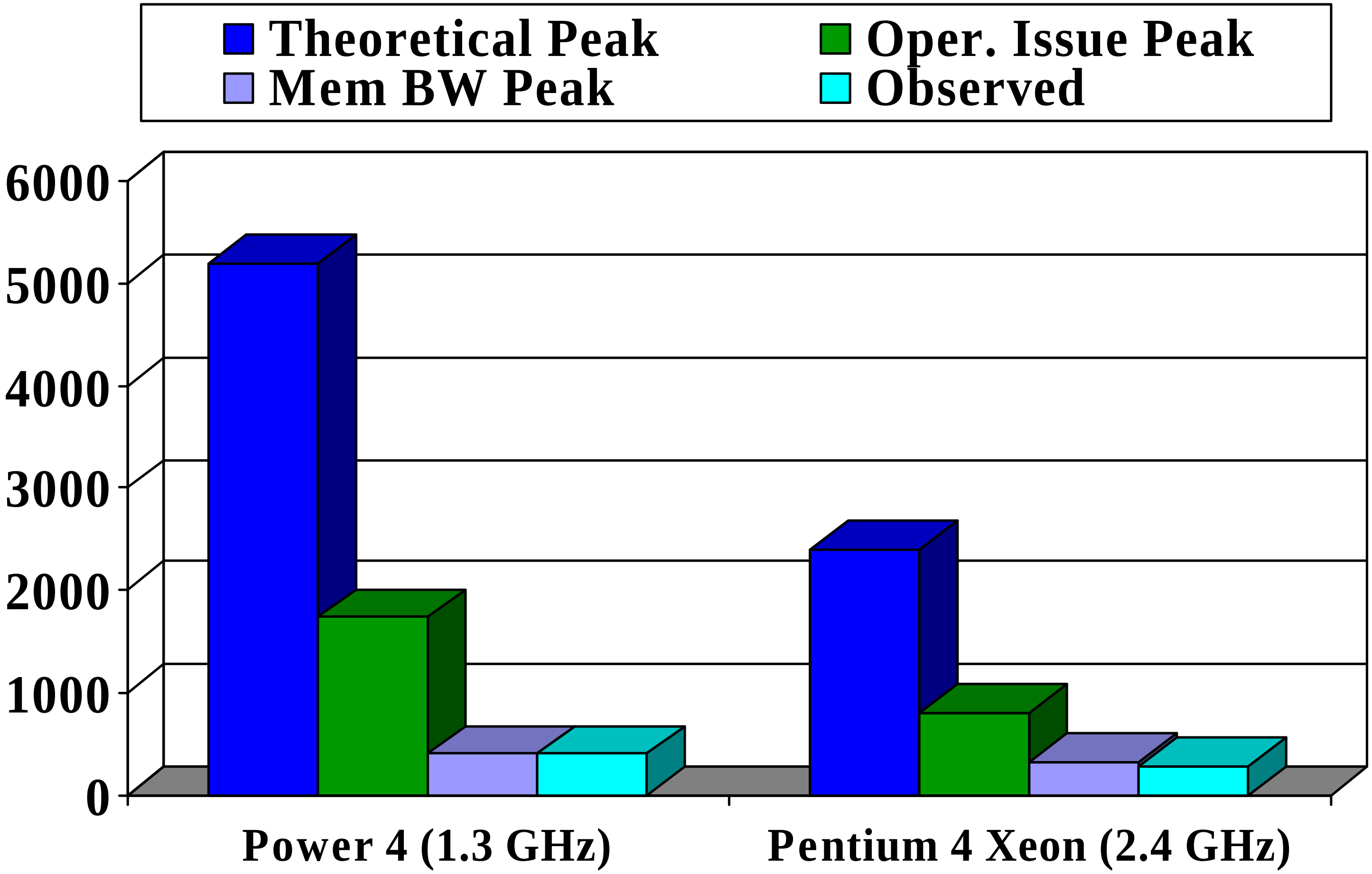
Simple Performance Analysis

- Memory motion:
  - nnz (sizeof(double) + sizeof(int)) + n (2*sizeof(double) + sizeof(int))
  - Assume a perfect cache (never load same data twice)
- Computation
  - nnz multiply-add (MA)
- Roughly 12 bytes per MA
- Typical workstation node can move 1-4 bytes/MA
  - *Maximum* performance is 8-33% of peak

Realistic Measures of Peak Performance

Sparse Matrix Vector Product

One vector, matrix size, m = 90,708, nonzero entries nz = 5,047,120



Thanks to Dinesh Kaushik;
ORNL and ANL for compute time

What About CPU-Bound Operations?

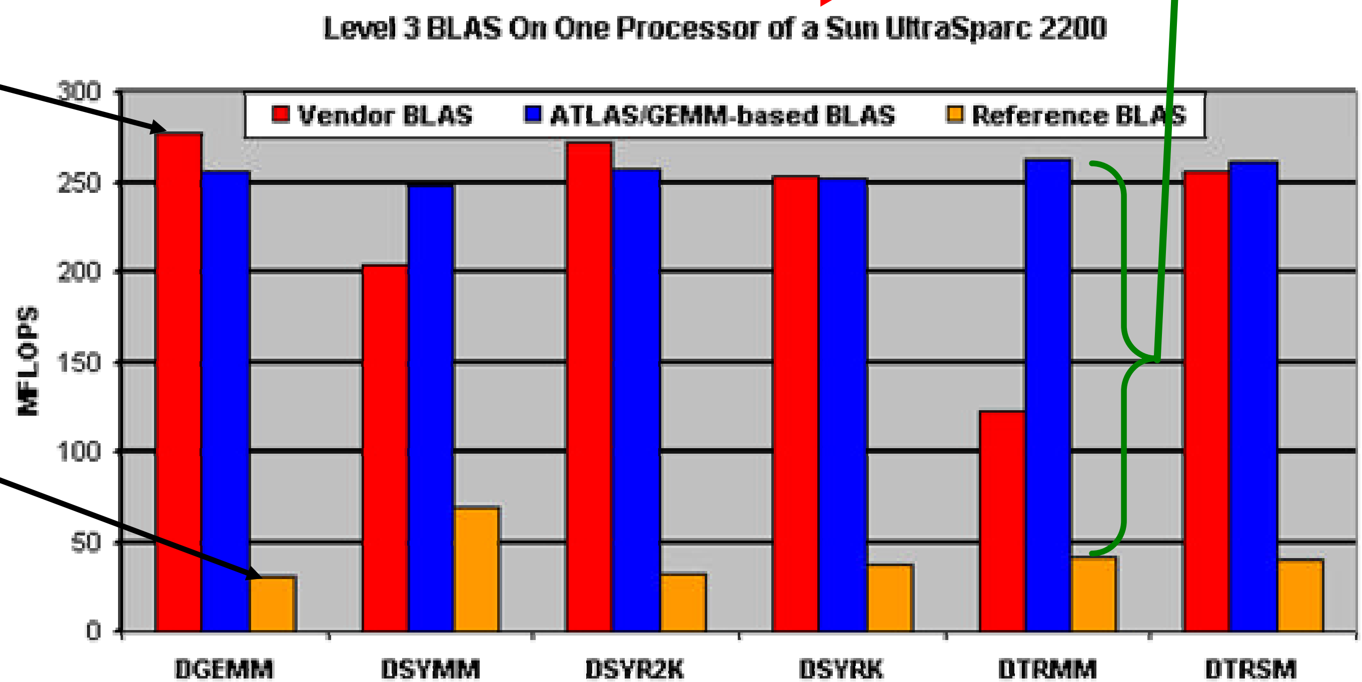
- Dense Matrix-Matrix Product
  - ♦ Most studied numerical program by compiler writers
  - ♦ Core of some important applications
  - ♦ More importantly, the core operation in High Performance Linpack
    - Benchmark used to “rate” the top 500 fastest systems
  - ♦ Should give optimal performance…

The Compiler Will Handle It (?)

Large gap between natural code and specialized code

Hand-tuned

Compiler



From Atlas

Enormous effort required to get good performance

Performance for Real Applications

- Dense matrix-matrix example shows that even for well-studied, compute-bound kernels, compiler-generated code achieves only a small fraction of available performance
  - ♦ "Fortran" code uses "natural" loops, i.e., what a user would write for most code
  - ♦ Others use multi-level blocking, careful instruction scheduling etc.
- Algorithm design must take into account the capabilities of the *system*, not just the hardware

What Performance Gap?

- Peak floating point rates do not predict performance
  - Performance models must look at *all* machine resources
- Even on simple codes, compilers are unable to deliver achievable performance
  - Code generation is *hard* and getting harder
  - Fully automatic, general purpose high performance code generation is a fantasy
- Achieving performance requires:
  - Algorithms designed for real hardware
  - Working with, not against, the programming models
  - Hardware with adequate performance
- Performance must be measured in terms of *science*
  - Floating point rates, neither peak nor achieved, are not good measures of performance